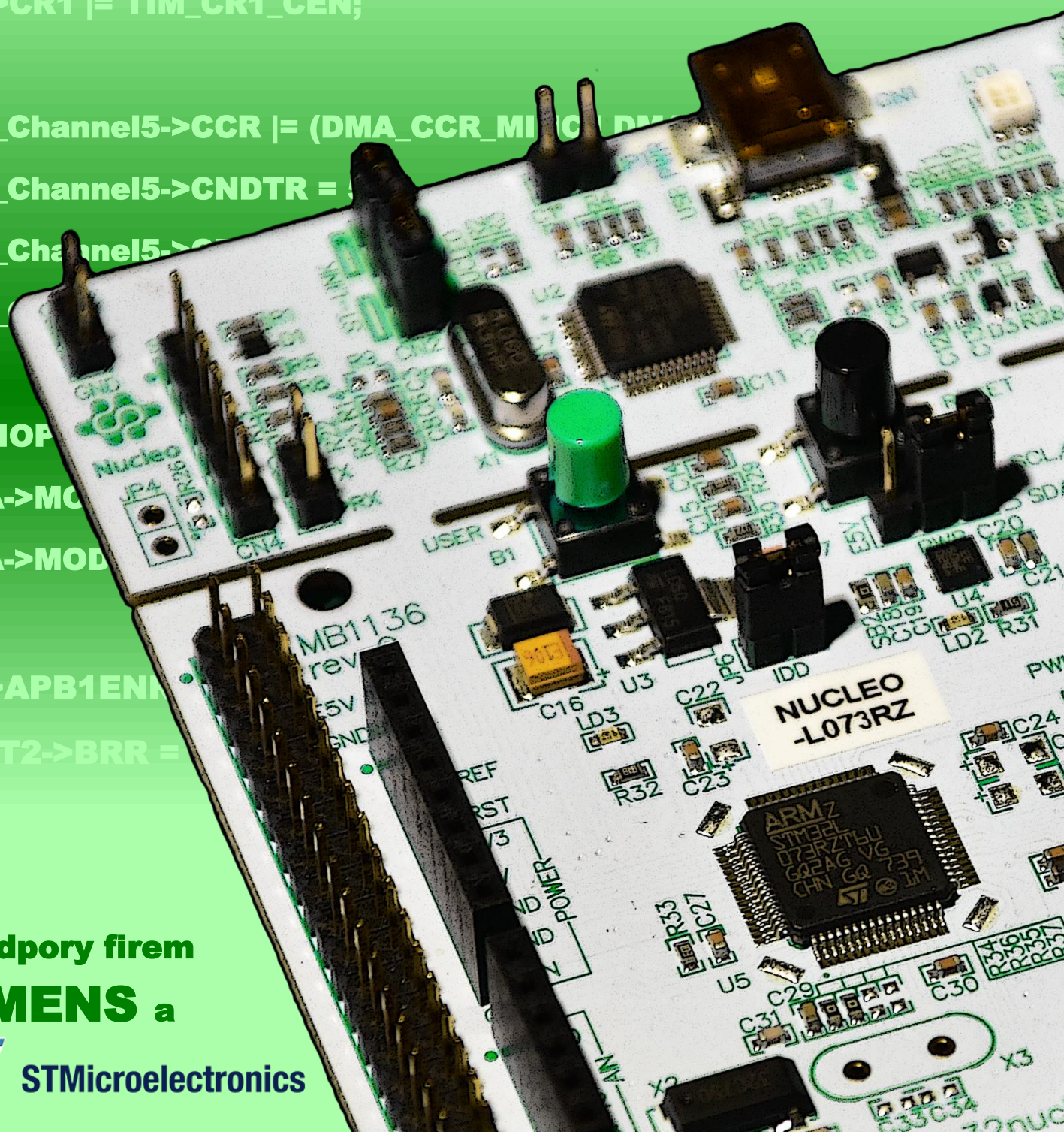
Programujeme STM32

bez knihoven

Ing. Vojtěch Skřivánek

```
TIM2->CR2 |= TIM_CR2_MMS_1;
TIM2->CR1 |= TIM_CR1_CEN;

DMA1_Channel5->CCR |= (DMA_CCR_MI
DMA1_Channel5->CNDTR =
DMA1_Cha nel5->
DMA1_

RCC->IOP
GPIOA->MO
GPIOA->MOD

RCC->APB1ENR
USART2->BRR =
```



Za podpory firem

SIEMENS a

STMicroelectronics

Poděkování

Děkuji firmám

SIEMENS

a

za podporu při psaní této knihy.

Obsah

Kapitola 6

Časovač

Časovač, zejména pokud se jedná a jeho rozšířenou verzi, je velice komplexní periferie.

Námi využitý časovač má 2 řídicí registry, které však skoro nevyužijeme. V praxi se pracuje hlavně s konfiguračními registry jako je registr nastavení předděličky, Autoreload registr, anebo porovnávací registr. Časovač má většinou více kanálů, proto jsou některé tyto konfigurační registry duplikovány pro každý kanál zvlášť. To samé platí pro některé bity v jednotlivých konfiguračních nebo stavových registrech.

6.1 Blikání pomocí časovače s přerušením

V prvním příkladu, který využívá časovače v základním režimu, si ukážeme, jak nastavit volně běžící časovač tak, aby každou sekundu vyvolal přerušení. V obsluze přerušení změníme logickou úroveň výstupního pinu, čímž dojde k zhasnutí či rozsvícení LED nacházející se na ***Nucleo*** desce s přesně definovaným intervalem jedné sekundy.

Navíc si v tomto příkladu ukážeme, jak můžeme aktivovat vysokorychlostní oscilátor integrovaný v čipu. Dále spustíme fázový závěs (***PLL - Phase-Locked Loop***), kterým budeme systémový hodinový signál upravovat na požadovanou frekvenci 32 MHz. To je maximální frekvence, se kterou kontroler dokáže pracovat.

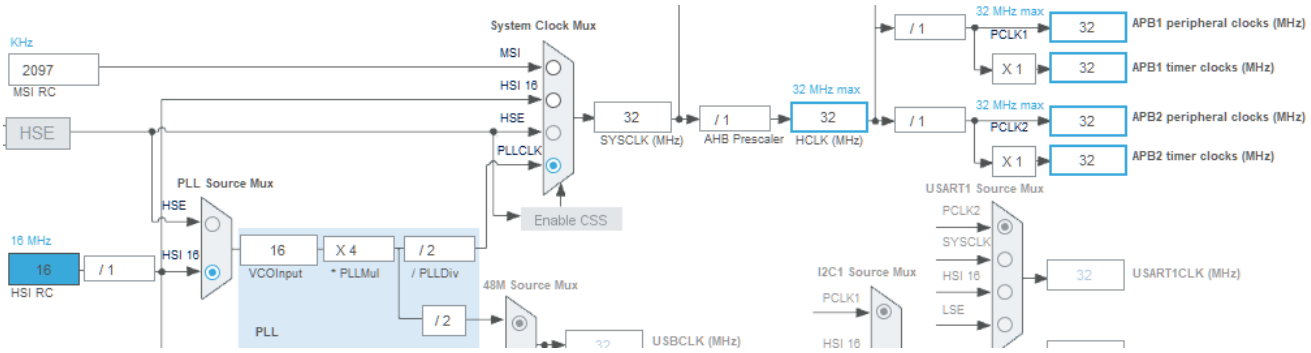
6.1.1 Nastavení periferií

6.1.1.1 RCC - hodinový signál

Nejprve si pro náš program připravíme zdroj hodinového signálu. V tomto příkladu budeme poprvé více pracovat s periferií ***RCC*** (***Reset and Clock Controller***).

Tato periferie umožňuje zapínat, volit, nastavovat a upravovat zdroje hodinového signálu.

Na obrázku z grafického konfigurátoru vidíme požadovaný výsledek. Chceme nastavit systémový hodinový signál na maximální frekvenci 32 MHz. Toho můžeme docílit tak, že aktivujeme interní vysokorychlostní oscilátor, který připojíme k fázovému závěsu. V něm kmitočet nejprve vynásobíme čtyřikrát a poté vydělíme dvěma. Výstup fázového závěsu pak zvolíme jako zdroj systémového hodinového signálu.



Následující program postupně provede přesně to, co jsme popsali výše.

```
void RCC_init()
{
        // zapne vnitrni vysokorychlostni oscilator (HSI - 16 MHz)
        RCC->CR |= RCC_CR_HSION;

        // cekej, dokud neni HSI pripraven
        while (!(RCC->CR & RCC_CR_HSIRDY));

        // nastavi nasobicku PLL na 4 (16 MHz * 4 = 64 MHz)
        // a delicku PLL na 2 (64 MHz / 2 = 32 MHz)
        RCC->CFGR |= (RCC_CFGR_PLLMUL4 | RCC_CFGR_PLLDIV2);

        // povoli PLL
        RCC->CR |= RCC_CR_PLLON;

        // cekej, dokud neni PLL pripraven
        while (!(RCC->CR & RCC_CR_PLLRDY));

        // nastavi cas pristupu do FLASH
        FLASH->ACR |= FLASH_ACR_LATENCY;

        // pouzije PLL jako zdroj systemovych hodin
        RCC->CFGR |= RCC_CFGR_SW_PLL;

        // cekej, dokud neni pouzit PLL jako systemove hodiny
        while(!(RCC->CFGR & RCC_CFGR_SWS_PLL));
}
```

Nejprve je aktivován vysokorychlostní oscilátor. Následující smyčka čeká, až bude ustálen. V následujícím kroku dojde k nastavení násobičky a děličky fázového závěsu, který se poté spustí. Opět se čeká na jeho přípravu.

Propojení vysokorychlostního oscilátoru a fázového závěsu je výchozí nastavení kontroleru. To však neplatí o nastavení zdroje systémového hodinového signálu, kterým je vícerychlostní vnitřní oscilátor. Proto je nutné toto nastavení změnit.

Než to ale provedeme, je nutné změnit dobu přístupu do paměti kontroleru. Jelikož po přepnutí zdroje signálu bude frekvence 32 MHz, bylo by čtení z paměti příliš rychlé. O tom se dočteme v příslušné kapitole manuálu kontroleru.

Table 12. Link between master clock power range and frequencies

Name	Power range	Maximum frequency (with 1 wait state)	Maximum frequency (without wait states)
Range 1	1.65 V - 1.95 V	32 MHz	16 MHz
Range 2	1.35 V - 1.65 V	16 MHz	8 MHz
Range 3	1.05 V - 1.35 V	4.2 MHz	4.2 MHz

Tabulka rozsahů napájecích napětí a maximálních frekvencí hodinových signálu [1]

Paměť by nedokázala procesoru při této frekvenci poskytnout data tak rychle. Z toho důvodu je nutné nastavit prodloužení čekací doby při čtení z paměti.

Po této změně, bez níž by kontroler nemohl správně fungovat, už stačí pouze zvolit fázový závěs jako zdroj systémového hodinového signálu a počkat, až k přepnutí dojde.

6.1.1.2 TIM

Nastavení časovače je v celku jednoduché. Nejprve jako vždy přivedeme do periferie hodinový signál.

Poté nastavíme konfigurační registry, děličku kmitočtu a Autoreload registr, tak, aby k přetečení došlo jednou za sekundu.

Povolíme přerušení při přetečení, a poté i linku přerušení jádra ARM, která je na časovač napojena.

```
void TIM2_init()
{
        // zapne hodinovy signal pro TIM2
        RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;

        // nastavi delicku pro TIM2 (32 MHz / 32000 = 1 kHz)
        TIM2->PSC = 31999U;

        // nastavi autoreload registr pro TIM2 (1 kHz / 1000 = 1 Hz)
        TIM2->ARR = 999U;

        // povoli preruseni pri preteceni (update)
        TIM2->DIER |= TIM_DIER_UIE;

        // povoli preruseni linky ARM jadra pro TIM2
        NVIC->ISER[0] |= (1U << TIM2_IRQn);
}
```

6.1.2 Funkce periferií

6.1.2.1 TIM - start

Funkce pro spuštění časovače jednoduše nastaví povolovací bit v jeho řídicím registru.

```
void TIM2_start()
{
        // zapne TIM2
        TIM2->CR1 |= TIM_CR1_CEN;
}
```

6.1.3 Obsluha přerušení

Obsluha přerušení je také velmi snadná.

Pokud je zdrojem přerušení přetečení časovače, dojde k přepnutí stavu LED a vynulování příznaku přerušení.

```
void TIM2_IRQHandler(void)
{
        // preruseni z duvodu preteceni TIM2
        if(((TIM2->SR & TIM_SR_UIF) == TIM_SR_UIF) &&
           ((TIM2->DIER & TIM_DIER_UIE) == TIM_DIER_UIE))
        {
                LED_prepni();

                // vynuluj priznak preruseni
                TIM2->SR &= ~(TIM_SR_UIF);
        }
}
```

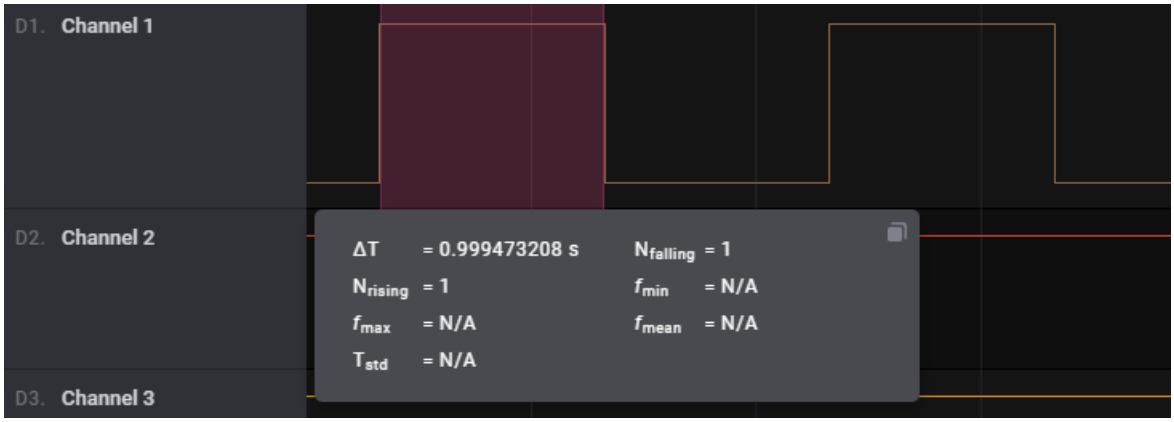
Teoreticky je v tomto příkladu podmínka zbytečná, jelikož jiný důvod přerušení není možný. Odstraněním podmínky kód zmenšíme a zrychlíme. Pro úplnost ji však do programu vložme, kdybychom se jej v budoucnu rozhodli rozšířit.

6.1.4 Tvorba programu

Celý program vypadá následovně. Nejprve nastavíme zdroj hodinového signálu. Poté připravíme funkcí LED na blikání. Následuje nastavení časovače a jeho spuštění. Nic víc není potřeba.

```
int main(void)
{
        RCC_init();

        LED_init();

        TIM2_init();

        TIM2_start();

        while(1);
}
```

Na následujícím obrázku je vidět, že doba svitu je skutečně jedna sekunda. Nepřesnost je dána tolerancí vnitřního oscilátoru.



6.1.5 Porovnání

Jelikož jsme v tomto příkladu pracovali s nastavením zdroje systémového hodinového signálu, není v porovnávacím programu s knihovnami odstraněna inicializační funkce. To má velký dopad na výsledky v oblasti velikosti kódu.

Bez optimalizace			
Kmitočet hodinového signálu = 32 MHz	**S knihovnami**	**Bez knihoven**	**Úspora [%]**
Velikost kódu [kB]	7,33	0,836	88,6
Obsluha přerušení přetečení [µs]	9,71	2,71	72,1
Optimalizace velikosti			
Velikost kódu [kB]	4,22	0,707	83,2
Obsluha přerušení přetečení [µs]	5,17	1,37	73,5
Optimalizace rychlosti			
Velikost kódu [kB]	4,99	0,668	86,6
Obsluha přerušení přetečení [µs]	4,79	1,50	68,7

Všimněme si také velkého rozdílu času obsluhy přerušení. Důvodem je komplexnost knihovní funkce, která dokáže reagovat na jakoukoliv příčinu přerušení.